Lecture 14: Time + Logical Clocks

(Based off slides by Rik Sarkar at University of Edinburgh)
Global time

- In practice, we act like there is a global notion of time
  
  *e.g., What time is it?*

- But, time is relative
  - Einstein showed speed of light constant for all observers
  - Leads to *Relativity of Simultaneity*

- Basically, impossible to tell if two events are simultaneous
  - If events are separated by space
  - *But*, if events are causally connected, can preserve order
Global time in systems

- For human-scale systems, these differences don’t matter
  - Rarely are we going near the speed of light

- But these do come to play in computing systems
  - Must consider relativity of time when designing systems
  - E.g., high-frequency trading systems
    - Need to be able to declare who bought first
  - Or, need to be able to merge multiple writes to single object
4 Outline

- Defining and measuring time
- Correcting clocks
- NTP
- Logical clocks
- Vector clocks
Historic clocks

- Our units of time date from the Sumerians in 2000BC
  - Sexagesimal system based on hand counting

- Humans used a variety of devices to measure time
  - Sundials
  - Astronomical clocks
  - Candle clocks
  - Hourglasses

- Mechanical clocks developed in medieval ages
  - Typically maintained by monks (church bell tower)
Electronic clocks

- First developed in 1920s
  - Uses carefully shaped quartz crystal
  - Pass current, counts oscillations

- Most oscillate at 32,768/sec
  - Easy to count in hardware
  - Small enough to fit (~4mm)

- Typical quartz clock quite accurate
  - Within 15 sec/30 days (6e-6)
  - Can achieve 1e-7 accuracy in controlled lab conditions
  - Not good enough for today’s applications
Atomic clocks

- Based on atomic physics
  - Cool atoms to near absolute zero
  - Bombard them with microwaves
  - Count transitions between energy levels

- Most accurate timekeeping devices today
  - Accurate to within $10^{-9}$ seconds per day
  - E.g., loses 1 second in 30 million years

- SI second now defined in terms of atomic oscillations
  - $9,192,631,770$ transitions of cesium-133 atom
Atomic clocks used to define a number of time standards

**TAI**: International Atomic Time

- Avg. of 200 atomic clocks, corrected for time dilation

- Essentially, a count of the number of seconds passed
Measuring real-world time

- Originally, each town defined noon locally
  - Point at which sun highest in the sky

- With growth of railroads, this became impractical
  - Continually have to re-set watches
  - Hard to set rail schedules

- Notion of “time zones” developed
  - Regions where wall-clock time is the same
  - Now, need to synchronize clocks
GMT, UT1, and UTC

- **GMT**: Greenwich Mean Time
  - Originally, mean solar time at 0º longitude
  - This isn’t really “noon” due to Earth’s axial tilt

- **UT1**: Universal Time
  - Modernized version of GMT
  - Based on rotation of Earth, ~86,400 seconds/day

- **UTC**: Universal Coordinated Time
  - UT1 + leap seconds
  - Minutes can have 59-61 seconds
  - Since 1972, 25 leap seconds have been introduced
11 Outline

- Defining and measuring time
- Correcting clocks
- NTP
- Logical clocks
- Vector clocks
Correctness

- What does it mean for a clock to be correct?
  - Relative to an “ideal” clock
  - *Clock skew* is magnitude, *clock drift* is difference in rates

- Say clock is correct within $p$ if
  
  $$(1-p)(t'-t) \leq H(t') - H(t) \leq (1+p)(t'-t)$$

  - $(t'-t)$ True length of interval
  - $H(t') - H(t)$ Measured length of interval
  - $(1-p)(t'-t)$ Smallest acceptable measurement
  - $(1+p)(t'-t)$ Largest acceptable measurement

- Monotonic property: $t < t' \Rightarrow H(t) < H(t')$
Monotonicity

- If a clock is running “slow” relative to real time
  - Can simply re-set the clock to real time
  - Doesn’t break monotonicity

- But, if a clock is running “fast”, what to do?
  - Re-setting the clock back breaks monotonicity
  - Imagine programming with the same time occurring twice

- Instead, “slow down” clock
  - Maintains monotonicity
Simple synchronization

- If we know message delay $T$
  - A sends current time $t$ to $B$, who sets time to $t+T$

- Typically, don’t know exact delay
  - May know range on delay $min < T < max$
  - $B$ can then set time to $t+(max-min)/2$
  - Clocks are then within $(max-min)/2$ of each other

- Can general this protocol to many clocks
  - Overall accuracy still $\sim(max-min)$

- But, don’t generally have any bound on delay
Cristian’s method

- No assumption of delay bound
- A sends request to B of current time
  - B responds with local time T
  - A measures RTT
  - A sets local time to $T + \text{RTT}/2$
- Assumes that delay is symmetric
  - Why?
- A can do this many times in a row, use overall min RTT
- Rough accuracy is $\text{RTT}/2 - \text{min}$, with overall min min
Defining and measuring time
Correcting clocks
NTP
Logical clocks
Vector clocks
Synchronizing in the real world

- *Network Time Protocol (NTP)* developed in 80s with goals
  - Keep machines synchronized to UTC
  - Deal with lengthy losses of connectivity
  - Enable clients to synchronized frequently (scalable)
  - Avoid security attacks

- NTP deployed widely today
  - Uses 64-bit value, epoch is 1/1/1900 (rollover in 2036)
  - LANs: Precision to 1ms
  - Internet: Precision to 10s of ms
NTP Hierarchy

- Based on hierarchy of accuracy, called *strata*
  - **Stratum 0**: High-precision atomic clocks
  - **Stratum 1**: Hosts directly connected to atomic clocks
  - **Stratum 2**: Hosts that run NTP with stratum 1 hosts
  - **Stratum 3**: Hosts that run NTP with stratum 2 hosts
  - ...

- **Stratum x hosts** often synch with other stratum x hosts
- Provides redundancy
NTP strata
NTP in practice

- Run on UDP port 123
- Most Internet hosts support NTP
- Accuracy on general Internet is ~10ms
  - Up to 1ms on local networks, ideal conditions
- Many networks run local NTP servers
  - E.g., time.ccs.neu.edu
- NTP has recently been a vector for DDoS attacks
  - Best practice is for servers to filter requests outside local network
Defining and measuring time
Correcting clocks
NTP
Logical clocks
Vector clocks
Logical ordering

- Goal: Be able to provide some synchronization of events
  - Recall, never able to do perfectly synchronize clocks

- How to deal with this fact in the real world?

- Create a new abstraction: *Logical ordering*
  - Remove real-world time from equation
  - Base ordering on causality

- Logical clocks are based on the simple principles:
  - 1. Events observed by a single process are ordered
  - 2. Any message must be sent before it is received
Example of logical ordering

- Each host can order all events it observes
  - B observes m1 received before m3 sent
- Can “interleave” timelines via messages
- Cannot make statement about all pairs of events
  - E.g., m5 send and m1 receive can’t be ordered
Formalize logical clocks via happened-before (→) relation

- If $e_1$ precedes $e_2$ on single host, then $e_1 \rightarrow e_2$
- If $e_1$ and $e_2$ and send/receive of message, then $e_1 \rightarrow e_2$
- $e_1 \rightarrow e_2$ and $e_2 \rightarrow e_3$, then $e_1 \rightarrow e_3$
- If neither $e_1 \rightarrow e_2$ nor $e_2 \rightarrow e_1$, then $e_1$ and $e_2$ are concurrent
- Say $e_1 \parallel e_2$
Limits of happened-before

- Cannot capture external events
  - Only considers message-passing; phone call?
  - Two events may be concurrent in our system

- If $e_1 \parallel e_2$, it does not imply causality
  - Potential causality is implied
  - E.g., process may receive message before unrelated event

- But, still pretty useful
  - How to implement logical ordering in a real system?
Logical clocks

- Lamport created way to create logical clock from ordering
- Define *logical clock* to be a monotonically increasing value
  - Numeric abstraction
  - Meaningless value by itself
- Each host $i$ maintains internal logical clock $L_i$
  - $L_i$ is incremented after each event
  - $L_i$ is piggy-backed on each message sent
  - Upon receipt of message with $t$
    - Set value to $\max(L_i, t) + 1$
Example of logical clocks

- For each event $e$, timestamp is longest chain of events that happened-before $e$
- Certain events cause “skipping” of clock
  - A’s clock skips from 1 to 5
No reverse implication

- We can observe that $e_1 \rightarrow e_2 \Rightarrow L(e_1) < L(e_2)$
  - If $e_1$ happened before $e_2$, then logical clocks ordered
- But the reverse is not true
  - $L(e_1) < L(e_2) \not\Rightarrow e_1 \rightarrow e_2$
- In example, $L(e) < L(b)$, but $e \not\leftrightarrow b$
  - In fact, $e$ concurrent with all but $f$
Defining and measuring time
Correcting clocks
NTP
Logical clocks
Vector clocks
Vector clocks

- Developed to overcome lack of reverse implication
  - Want $L(e_1) < L(e_2) \Rightarrow e_1 \rightarrow e_2$

- Processes keep local vector clock $V_i$
  - Array of logical clocks of length $N$ (# processes)
  - Initially [0,0, ... 0]

- Similar update procedure to logical clocks
  - $V_i[i]$ is incremented after each event
  - $V_i$ is piggy-backed on each message sent
  - Upon receipt of message with vector clock $V_k$
    - $V_i[x] = \max(V_i[x], V_k[x]) + 1$, for all $x$
Example of vector clocks

Invariant:

\[ V_{ij} \] is the number of events in process \( P_j \) that happened before the current state of process \( P_i \)
Comparing vector timestamps

- Given two vector timestamps $V_i$ and $V_j$
  - $V_i = V_j$ iff $V_i[x] = V_j[x]$ for all $x$
  - $V_i < V_j$ iff $V_i[x] < V_j[x]$ for all $x$

- For example, $(2,4,1) < (3,5,9)$

- But, other pairs incomparable
  - E.g., $(2,4,1)$ and $(3,1,7)$

- As with logical clocks $e_1 \rightarrow e_2 \Rightarrow L(e_1) < L(e_2)$

- And also $L(e_1) < L(e_2) \Rightarrow e_1 \rightarrow e_2$
Vector vs. logical clocks

- Vector clocks augment logical clocks
  - Use generalization of same mechanism

- Cost: Larger messages, more complexity
  - Often don’t know total number of processes

- But, with both can say when certain events happened before each other

- Also, can extent vector clocks to \textit{matrix clocks}
  - Your logical clock + others’